ELSEVIER Journal of Computational and Applied Mathematics 66 (1996) 267–278

JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS

Parallel point- and domain-oriented multilevel methods for elliptic PDE's on workstation networks

Michael Griebel, Tilman Neunhoeffer*

Institut für Informatik, Technische Universität München, D-80290 München, Germany

Received 10 October 1994; revised 17 March 1995

Abstract

We present new multilevel methods for the solution of linear elliptic PDEs. They show the same convergence behavior as conventional multigrid methods but possess interesting properties with respect to parallelization. Regarding communication, the number of setup steps is only dependent on the number of processors and not on the number of levels like for parallelized multigrid methods. This makes our new methods perfectly suited for parallel computing systems with relatively dominant communication setup like for example clusters of workstations.

Keywords: Partial differential equations; Multilevel methods; Semidefinite system; Iterative methods; Domain decomposition; Parallelization

AMS classifications: 65F10; 65N22; 65N55; 65Y05; 65Y10

1. Introduction

Recently, a new concept for the development of multigrid and BPX-like multilevel algorithms for the solution of elliptic PDEs had been presented (see [4, 5]). There, instead of a basis approach on the finest grid and the acceleration of the basic iteration by a MG-coarse grid correction or a BPX-type preconditioner, a generating system was used. Its degrees of freedom are associated to the nodal basis functions of *all* levels under consideration and thus allow a nonunique levelwise decomposed representation of the solution. Now, the Galerkin approach leads to a semidefinite linear system with unknowns on all levels. The generalized condition number (i.e. the quotient of the largest and smallest *nonvanishing* eigenvalue) of this system is of the order O(1). Its solution is nonunique but in some sense equivalent to the unique solution of the standard problem on the finest grid.

* Corresponding author. E-mail: neunhoef@informatik.tu-muenchen.de.

SSDI 0377-0427(95)00195-6

It turns out, that traditional iterative methods (Gauss–Seidel, conjugate gradients) for the semidefinite system are equivalent to modern multilevel methods (multigrid, BPX), if we use a levelwise ordering of the unknowns. But the semidefinite system also allows us to abandon the level-oriented view. For example, we can group together all unknowns which belong to different levels but are associated to one grid point. This results in point-oriented methods and can be considered as a point-block technique (see [3, 5]). Furthermore, we can group together all unknowns which belong to different levels but are associated to the same part of the domain. There, the subgroups are formed by recursive substructuring of the overall domain (resulting in subdomains and separators). Thus, we obtain some sort of simple domain decomposition methods, which exhibit MG-type convergence properties.

As the point- and domain-oriented methods allow directly an interpretation in terms of domain decomposition, their *parallelization* is straightforward. In contrast to the parallelization of a multilevel method where communication has to take place on *all* levels to maintain good convergence rates, our approach needs substantially less communication steps and therefore less setup time, due to its domain decomposition qualities. In this sense, our new methods are superior to other parallel multigrid and multilevel methods. In addition, they are well suited for networks of workstations which are configurated as binary trees. Here, the subdomains on the finest level of the above-mentioned substructuring process are treated in the leaves of the tree and the separators are treated in the nodes of the tree. Thus, data exchange can take place in parallel on each level of the tree.

We discuss the parallelization properties of these new methods and present results obtained on a tree-structured network of workstations.

2. The semidefinite system

Consider a second-order partial differential equation with linear, symmetric and elliptic operator $Lu = f$ in $\Omega := (0,1)^2$ with Dirichlet boundary conditions and associated weak formulation $a(u,v) = f(v)$, $\forall v \in V$. For the discretization on some grid Ω_k with uniform mesh width $h_k = 2^{-k}$, usually, a nodal basis $B_k = \{\phi_i^{(k)}, i = 1, \ldots, n_k\}$, $n_k = (2^k - 1)^2$, is used. The bilinear basis functions $\phi_i^{(k)}$ are defined by $\phi_i^{(k)}(x_j) = \delta_{ij}$, $x_j \in \Omega_k$, $i, j = 1, \ldots, n_k$, and they span the corresponding space $V_k = \operatorname{span}\{\phi_i^{(k)}, i = 1, \ldots, n_k\}$. Here, n_k denotes the number of interior grid points and thus the dimension of V_k. Any function $u \in V_k$ can be denoted by

$$u = \sum_{i=1}^{n_k} u_{k,i} \cdot \phi_i^{(k)},$$

with corresponding coefficient vector $u_k^B = (u_{k,i})_{i=1,\ldots,n_k}$ of nodal values, that is $u(x_i) = u_{k,i}$. Now, the Galerkin approach leads to the linear system

$$L_k^B u_k^B = f_k^B \tag{1}$$

with the vector u_k^B of unknowns.

It is well known that the condition number of L_k^B behaves like $O(h_k^{-2})$. So, the more unknowns we have, the more iteration steps an iterative method for the solution of (1) needs. This problem can be overcome by multilevel methods.

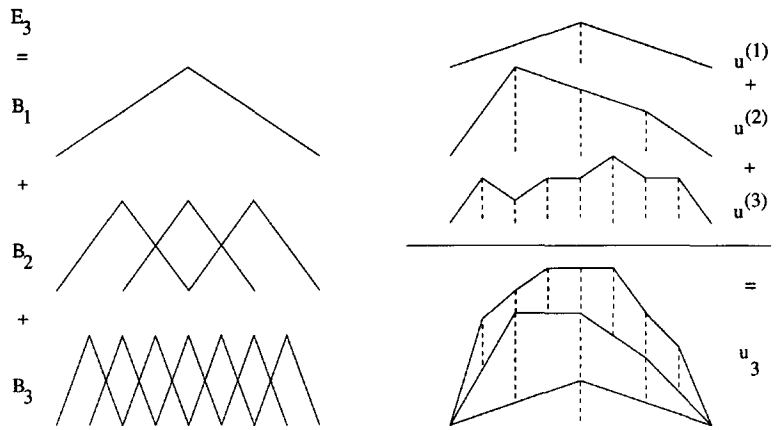

Fig. 1. Generating system E_3 (left) and multilevel representation of u_3 by E_3 (right) in 1D.

Here a sequence $\Omega_1 \subset \Omega_2 \subset \cdots \subset \Omega_k$ of grids, with an associated sequence $B_1, B_2, \ldots, B_k$ of nodal bases and corresponding spaces $V_1 \subset V_2 \subset \cdots \subset V_k$ with dimensions $n_1, n_2, \ldots, n_k$ is employed. Inspired by that, we now will directly use the generating system

$$E_k := B_1 \cup B_2 \cup \cdots \cup B_k = \bigcup_{l=1}^{k} B_l$$

for the representation of functions in V_k and for the discretization process. Compare also [4, 5]. Since E_k is only a generating system and not a basis for V_k, the representation of any function $u \in V_k$ by

$$u = \sum_{l=1}^{k} \sum_{i=1}^{n_l} u_{l,i} \cdot \phi_i^{(l)},$$

with the enlarged vector $u_k^E = (u_1^{B^T}, u_2^{B^T}, \ldots, u_k^{B^T})^T$ is no more unique.

For the 1D case, Fig. 1 shows the functions contained in E_3 and an example of a multilevel representation of a function $u_3 \in V_3$.

Now, we use the generating system E_k directly in the Galerkin discretization process. Then, we obtain the enlarged linear system

$$L_k^E u_k^E = f_k^E, \tag{2}$$

with a semidefinite matrix L_k^E where for $i_1 = 1, \ldots, n_{l_1}$, $i_2 = 1, \ldots, n_{l_2}$ and $l_1, l_2 = 1, \ldots, k$

$$(L_k^E)_{i_1, i_2, l_1, l_2} = a(\phi_{i_1}^{(l_1)}, \phi_{i_2}^{(l_2)}) \quad \text{and} \quad (f_k^E)_{i_2, l_2} = f(\phi_{i_2}^{(l_2)}).$$

This linear system is of size $n_k^E = \sum_{l=1}^{k} n_l$, which is in 1D about 2 times, in 2D about $\frac{4}{3}$ times and in 3D about $\frac{8}{7}$ times larger than n_k, i.e. the size of (1).

Assuming a level-oriented ordering of the unknowns, we obtain the following structure for L_k^E (here for the simple example of $k = 3$):

$$L_k^E = \begin{pmatrix} R_3^1 L_3^B P_1^3 & R_3^1 L_3^B P_2^3 & R_3^1 L_3^B \\ R_3^2 L_3^B P_1^3 & R_3^2 L_3^B P_2^3 & R_3^2 L_3^B \\ L_3^B P_1^3 & L_3^B P_2^3 & L_3^B \end{pmatrix} = \begin{pmatrix} R_3^1 \\ R_3^2 \\ I_3 \end{pmatrix} \cdot L_3^B \cdot (P_1^3 \;\; P_2^3 \;\; I_3),$$

where $R_i^j = P_j^{i^T}$ and P_j^i denotes the interpolation/prolongation from V_j to V_i, $j < i$, and I_i denotes the identity in V_i. Note that $P_j^i = \prod_{q=1}^{i-j} P_{i-q}^{i-q+1}$. Thus, we see that with help of the matrix

$$S_k := (P_1^k \;\; P_2^k \;\; \cdots \;\; P_{k-1}^k \;\; I_k),$$

our enlarged system (2) can be written as

$$S_k^T L_k^B S_k u_k^E = S_k^T f_k^B. \tag{3}$$

Then, we see that the discrete Galerkin operators $L_i^B = R_k^l L_k^B P_l^k$, $l = 1, \ldots, k$, i.e. the stiffness matrices of each level of discretization, are contained as diagonal blocks. The couplings between different levels are contained in the outer diagonal blocks.

Note that our enlarged system is consistent, i.e. $\operatorname{rank}(L_k^E) = \operatorname{rank}(L_k^E, f_k^E)$, and therefore solvable. However, there exists not only one unique solution but many different solutions due to the semidefiniteness of L_k^E. But, since the unique solution u_k^B of (1) can be gained from *any* solution u_k^E of (2) by $u_k^B = S_k u_k^E$, the idea is now to apply S_k to some solution u_k^E of (2) produced by any iterative method which converges to a value only dependent on the first iterate. This will be studied in the following sections.

3. Level-oriented methods

In the previous example we employed a levelwise ordering of the unknowns u_k^E that resulted in a level-block partitioning of the matrix L_k^E and the system (2) and was associated to the splitting $V_k = \sum_{l=1}^k V_l = \sum_{l=1}^k \sum_{i=1}^{n_l} V_{l,x_i}$, where $V_{l,x_i} = \operatorname{span}\{\phi_i^{(l)}\}$.

It can be seen easily that traditional iterative methods for (2) are equivalent to modern multilevel methods for (1), cf. [4, 5]. For instance, the simple Jacobi-preconditioner for (2) resembles just the BPX-preconditioner [1] for (1). The BPX-preconditioner can be written as $BPX_k = S_k (D_k^E)^{-1} S_k^T$, where $D_k^E = \operatorname{diag}(L_k^E)$. Now, if we define the generalized condition number κ of a positive semidefinite matrix to be the quotient of the largest and nonvanishing smallest eigenvalue, we obtain directly

$$\kappa(BPX_k L_k^B) = \kappa(S_k (D_k^E)^{-1} S_k^T L_k^B) = \kappa((D_k^E)^{-1} S_k^T L_k^B S_k) = \kappa((D_k^E)^{-1} L_k^E)$$

and since $\kappa(BPX_k L_k^B) = O(1)$ (cf. [11–14]) we have

$$\kappa((D_k^E)^{-1} L_k^E) = O(1). \tag{4}$$

Thus, the Jacobi-preconditioned CG-method for (2) converges to some solution within a number of iterations that is independent of k.

Furthermore, we can consider Gauss–Seidel-type methods for (2). They are equivalent to multigrid methods with Gauss–Seidel smoother, cf. [4, 5]. For example, the plain Gauss–Seidel iteration on (2) with levelwise ordering $l = 1, \ldots, k$ resembles just the multigrid (0, 1)-V-cycle with one post-smoothing step by Gauss–Seidel. The symmetric Gauss–Seidel iteration corresponds to the (1, 1)-V-cycle. Here, an outer iteration switches from level to level and an inner iteration operates on the specific grids.

The convergence rate of the Gauss–Seidel iteration on (2) can be estimated by

$$\rho = 1 - O(1)$$

(see [13]). For further results, compare also [6, 8].

Interestingly, this holds for *all* possible Gauss–Seidel traversal orderings, cf. [5]. Therefore, we obtain a k-independent convergence rate not only for the Gauss–Seidel method for (2) with some levelwise traversal ordering that corresponds to a multigrid method, but for *any other* traversal ordering as well. This will be exploited in the sequel.

4. Point- and domain-oriented methods

Now, we will change the traversal ordering to obtain new multilevel methods. Therefore, we partition the unknowns of the semidefinite system into groups and perform a symmetric block Gauss–Seidel iteration on the associated block-partitioned system. The resulting block-systems are treated by an inner iteration (e.g. Gauss–Seidel) or a direct solver.

For example, we can use a point-oriented ordering. Here we group together all unknowns that belong to one gridpoint but which are related to different levels. We obtain the following splitting of V_k:

$$V_k = \sum_{x \in \mathcal{N}_k} \sum_{l : x \in \mathcal{N}_l} V_{l,x},$$

where $\mathcal{N}_l$ is the set of grid points of grid Ω_l. Then, the block Gauss–Seidel iteration switches from grid point to grid point and the unknowns that belong to one grid point are relaxed in an inner iteration, either by a direct solver or by an iterative method.

Another possibility is a domain-oriented ordering. Here we may allow an arbitrary domain decomposition of Ω into nonoverlapping subdomains. Then we group together all the unknowns of the semidefinite system that are associated to basis functions whose center points are situated in the respective subdomain.

A decomposition which is especially well suited for parallelization is a decomposition into subdomains Ω_1 and Ω_2 and a separator S along a middle line, such that the functions of the generating system E_k related to Ω_1 and Ω_2 have disjoint supports. So the unknowns belonging to Ω_1 are not dependent on the unknowns of Ω_2 and the two subdomains can be treated in parallel. A simple 1D example is shown in Fig. 2.

Using a nested dissection strategy [2], each of the subdomains Ω_* can be divided again in a separator S_* and the two smaller subdomains Ω_{*1} and Ω_{*2} in a recursive substructuring process. In 2D, this can either be done using parallel lines or lines with alternating directions (cf. Fig. 3).

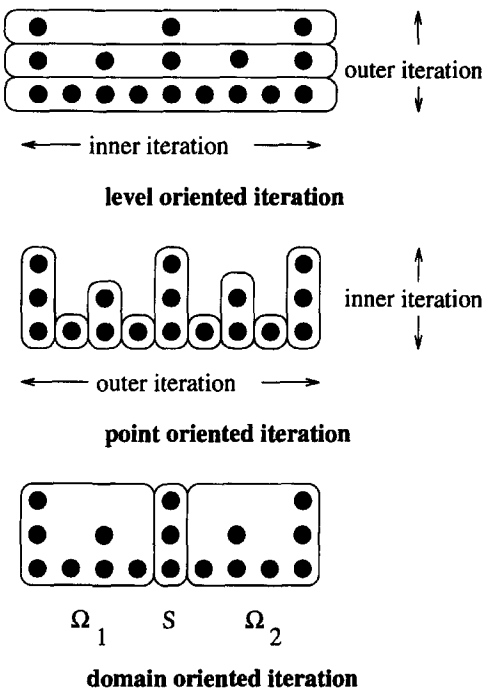

Fig. 2. Different traversal orderings.

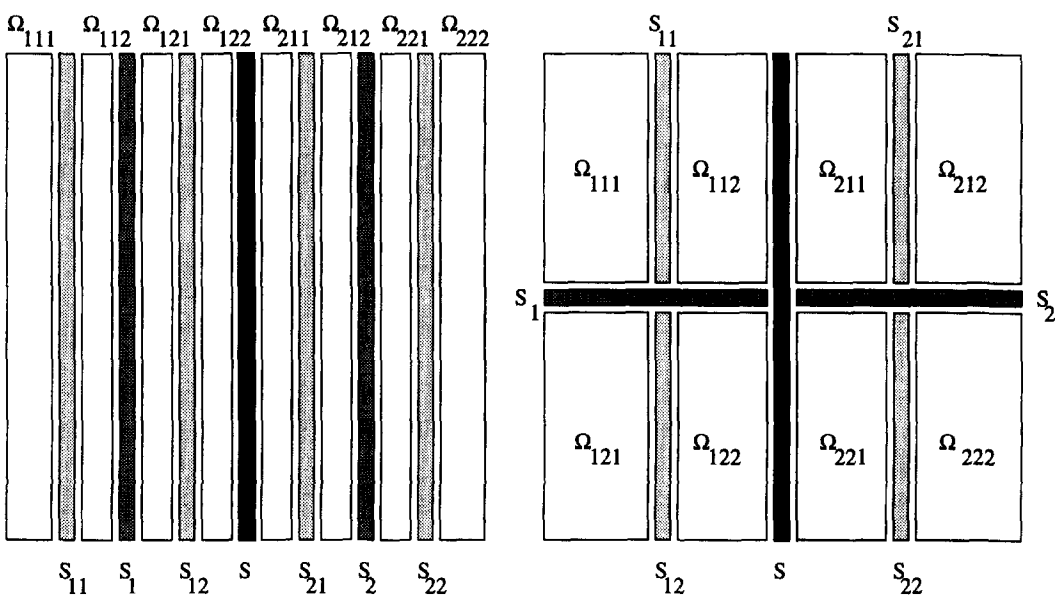

Fig. 3. Stripe- and box-wise nested dissection decompositions of Ω.

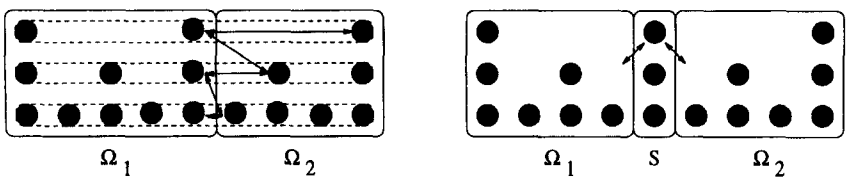

Fig. 4. Communication (↔) for the level- and domain-oriented ordering with two subdomains in 1D.

Thus, we end up with a stripe- or box-wise decomposition of Ω and we have a simple domain decomposition method which exhibits multigrid-like convergence properties.

Note that, for all cases, it is not necessary to assemble the matrix L_k^E and the right-hand side f_k^E explicitly. Similar to (3) it is possible to use prolongation and restriction operators and the fine grid discretizations L_k, f_k to express L_k^E and f_k^E in a certain product form. Furthermore, by storing and updating certain parts of the current residual, it is possible to implement the point- and domain-oriented Gauss–Seidel methods to need $O(n_k^E) = O(n_k)$ operations per iteration step, only. Especially for the point- and domain-oriented block Gauss–Seidel methods, this is quite technical. A description of implementational details will be given in [7]. Compared with the traditional multigrid correction scheme, the point-oriented method needs about 6% more operations and for the domain-oriented approach the number of operations is multiplied by a factor between 1 and 1.5, depending of the number of subdomains.

Although the convergence rate is independent of the number of levels k for all traversal orderings, the rates for point- and domain-oriented methods are slightly worse than for levelwise orderings. In numerical experiments, we obtained convergence factors of 0.1–0.3 for Poisson's equation.

Thus, we can pose the question whether we need the point- and domain-oriented method at all. The answer gets clear if we consider the parallelization properties of the different methods on most presently available MIMD computers and especially on networks of workstations. There, the time necessary to setup communication is often relatively large. Thus, it is more advantageous to exchange a larger amount of data collectively in one step than to exchange only fractions of the data in many different steps. Otherwise it can happen that the overall execution time is dominated mainly by the setup time.

In this respect, the level- and the point-/domain-oriented methods are different. For the parallel version of the level-oriented method, we need communication on each level. So the number of communication setups is of the order $O(k)$, where k is the number of levels. For details on the parallelization of multigrid methods, see [9, 10].

On the other hand, for the domain-oriented method with two subdomains, we can assign each of the subdomains and the separator to one process. Then, within a symmetric Gauss–Seidel iteration step we only need two communication steps between the subdomains and the separator, not dependent on the number of levels: one in the forward and one in the backward step. The communication steps needed for the level- and domain-oriented ordering are illustrated in Fig. 4.

If we use more than two subdomains, gained by a nested dissection process, we obtain a binary tree-like structure of processes. Here the subdomains are treated in the leaves of the tree and the

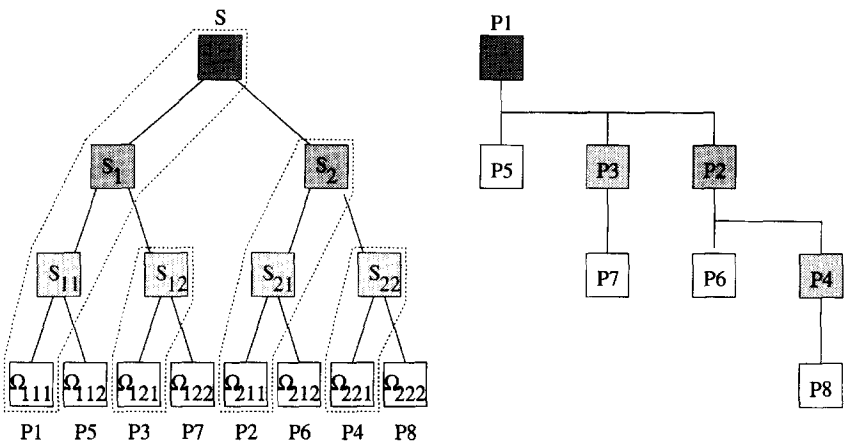

Fig. 5. Process tree (left) and network configuration (right).

separators are treated in the nodes. For example the unknowns related to the line in the middle of the overall domain (the separator S) are treated in the root of the tree, cf. Fig. 5, left.

Thus, processes on one level of the tree can operate in parallel whereas processes on different levels must work sequentially. Then, we can map several processes of different levels to one processor, namely each process and its left sons (cf. Fig. 5, right). The number of processors used is only half of the number of processes. The number of communication steps is of the order $O(\log_2 P)$, where P is the number of processors and communication between processes on adjacent levels can be executed in parallel. Nevertheless, some processors are idle if the separators are processed.

5. Results

We now report on the results of our numerical experiments. We consider the model problem

$$-\Delta u = f \quad \text{in } \Omega = (0,1)^2,$$

with Dirichlet boundary conditions. Note that this simple Poisson problem is a hard test case when it comes to parallelization, since the ratio of arithmetic operations to communication costs is worse than for all other second-order elliptic PDEs.

We implemented a mixed point- and domain-oriented symmetric Gauss–Seidel iteration on the semidefinite system. In an outer iteration, we switch from level to level in the process tree (see Fig. 5), first from the leaves to the root and then vice versa. On each level the different subdomains, respectively, separators are treated in parallel using one Gauss–Seidel step with a point-oriented ordering of the unknowns. The subdomains were produced by a nested dissection strategy which is based on the alternating lines approach as shown on the right part of Fig. 3.

We run it in parallel on a network of HP 9000/720 workstations that are equipped with *two* Ethernet cards. Thus, it is possible, to build tree-like configurations, see Fig. 5. Communication can take place in parallel on the Ethernet connections of the different levels of the processor tree. The

Table 1
Setup time S for $k = 3, \ldots, 11$

P	1	2	4	8	16
S	0.000	0.022–0.024	0.044–0.047	0.067–0.075	0.090–0.097

code is written in C and uses the PVM communication system. For compilation, we used the O option.

In the following tables we show the time in seconds, used for computation, communication and setup in one iteration step of the symmetric Gauss–Seidel relaxation performed on the semidefinite system. We considered it for $k = 3, \ldots, 11$ levels and $P = 1, \ldots, 16$ processors.

First, we measured the communication setup time S [1] for one symmetric Gauss–Seidel iteration step. The results are shown in Table 1. They are independent of the number k of levels but only depend on the number P of processors. We obtained $S \approx 0.023 \log_2(P)$.

Secondly, we measured the communication time C, the computation time W and the total time T for one symmetric Gauss–Seidel iteration of the overall algorithm, i.e. $T = S + C + W$. The results are shown in Table 2. Note that the missing values for $k = 11$, $P = 1, 2, 4, 8$ are not given since their computation involved paging and swapping effects.

Regarding the communication time it can be seen that C behaves for fixed k like $O(\log_2 P)$. On the other hand, it depends of the size of the largest separator. So, for fixed P we must obtain $C = O(2^k)$. A short analysis of our data shows that

$$C \approx \log_2(P) \cdot (0.013 + 4 \cdot 10^{-5} \cdot 2^k).$$

But this only holds for low numbers of P ($P < 32$). From theory [5], we know that the amount of data to be communicated sequentially does not depend on P for sufficiently large values of P since some sort of geometric series develops. We then would have $C \approx 0.013 \log_2(P) + 4 \cdot 10^{-5} \cdot 2^k \cdot 6.75$ but this cannot be seen from our table.

For the computation time we have $W = c_1 \cdot 2^{2k}/P + c_2 \cdot 2^{k+1}$ where the first term relates to the time spent for the computation of the subdomains and the second term to the time spent for the separators. The constants c_1 and c_2 are in the same range. For $k > \log_2(P) + 4$ the first term dominates and we can derive from our data that

$$W \approx 2.5 \cdot 10^{-5} \cdot 2^{2k}/P.$$

Furthermore, we see that W nicely scales with $W^{k,P} = W^{k+1,4P}$ for $k \geqslant 7$. However, since the HP 9000/720 is a cache-based RISC architecture, there might be an effect of out-of-cache computations onto W but this cannot be seen clearly in our measurements.

[1] Here, we measured the time needed to send only one data in each communication step without communication of the whole data and the computation of the new iterates and residuals. The initialization time, i.e. the time necessary to distribute the code and the processes on the network and to allocate necessary data at the beginning of the computation is not contained.

Table 2
Communication time C, computation time W and total time T for the domain-oriented algorithm and expected setup times for parallelized multigrid, using a Block-Jacobi (BJ), Red–Black–Gauss–Seidel (RB) or Four-Color–Gauss–Seidel (FC) smoother

	P	1	2	4	8	16	BJ	RB	FC
$k = 3$	C	0.000	0.013	0.026	0.042	0.059			
	W	<0.001	<0.001	<0.001	<0.001	<0.001			
	T	<0.001	0.036	0.070	0.109	0.149	0.110	0.220	0.440
$k = 4$	C	0.000	0.012	0.025	0.044	0.061			
	W	0.003	0.002	0.002	0.001	0.001			
	T	0.003	0.036	0.072	0.112	0.152	0.154	0.308	0.616
$k = 5$	C	0.000	0.012	0.026	0.046	0.064			
	W	0.013	0.008	0.005	0.002	0.001			
	T	0.013	0.042	0.077	0.116	0.155	0.198	0.396	0.792
$k = 6$	C	0.000	0.013	0.027	0.050	0.068			
	W	0.062	0.030	0.016	0.012	0.011			
	T	0.062	0.066	0.091	0.131	0.170	0.242	0.484	0.968
$k = 7$	C	0.000	0.016	0.034	0.057	0.073			
	W	0.356	0.160	0.077	0.038	0.035			
	T	0.356	0.200	0.157	0.166	0.192	0.286	0.572	1.144
$k = 8$	C	0.000	0.022	0.042	0.065	0.104			
	W	1.599	0.780	0.376	0.180	0.095			
	T	1.599	0.825	0.466	0.321	0.293	0.330	0.660	1.320
$k = 9$	C	0.000	0.027	0.067	0.102	0.127			
	W	6.478	3.230	1.620	0.812	0.404			
	T	6.478	3.291	1.734	0.987	0.627	0.374	0.748	1.596
$k = 10$	C	0.000	0.068	0.118	0.173	0.217			
	W	26.111	13.064	6.538	3.304	1.696			
	T	26.111	13.156	6.703	3.551	2.001	0.418	0.836	1.672
$k = 11$	C					0.392			
	W					6.657			
	T					7.147	0.231	0.924	1.848

Altogether, the part of the execution time which only depends on the number of communication steps is quite dominating. For example, even in the case $P = 16$, $k = 8$ the setup time S is still as large as the computation time W.

For comparison purposes, we show the setup times necessary for conventional parallel multigrid methods ((1, 1)-V-cycle) on the right-hand side of Table 2. Here, we must consider the necessary communication steps for the smoothing, the restriction and the prolongation operators. First, we discuss different smoothing operators. If we use a Block–Jacobi (BJ) smoother, where the blocks contain all unknowns related to one subdomain and one level, and a Gauss–Seidel iteration in each block, communication is needed only after each smoothing step. Therefore, one iteration cycle involves *at least* $2k - 1$ communication steps for the smoothing operators on the different levels.

Table 3
Efficiencies E in %

P	2	4	8	16
$k = 7$	89.1	56.7	26.8	11.7
$k = 8$	97.0	85.7	62.3	34.1
$k = 9$	98.4	93.4	82.1	64.6
$k = 10$	99.2	97.4	92.0	81.2
$k = 11$				91.3

However, the convergence rates deteriorate with a growing number of subdomains. Better convergence results can be obtained using a Gauss–Seidel smoother with a Red–Black (RB) or a Four-Color (FC) ordering of the unknowns for a five-point or nine-point discretization of the Poisson equation, respectively. Here, we need data communication after the relaxation of all points related to the same color and the same level. Thus, one iteration cycle involves *at least* $4k - 2$ (RB) or $8k - 4$ (FC) communication steps for smoothing. For the prolongation and restriction operators, we either need further setup steps or a larger overlap of the subdomains. Therefore, we take for our considerations only the setup steps of the smoother into account.[2]

If we multiply the numbers of communication steps with the setup time for one bidirectional communication step of 0.022 s, which we obtained in our experiments, we can expect *at least* the setup times noted in Table 2 for standard parallel multigrid. We can see, that for $P = 16$ and $k \leqslant 9$, only the setup times for standard multigrid algorithms are already in the range of the total time of our domain-oriented algorithm. Thus, if we add the time needed for computation and communication, we see the superiority of our domain oriented approach on parallel systems with high setup times.

In Table 3 the efficiencies are given for $k = 7, \ldots, 11$. Here, the efficiency E is defined by the ratio

$$E(P) = \frac{T(1)}{T(P) \cdot P},$$

where $T(P)$ denotes the time needed for the calculation on P processors. For $k = 11$, $T(1)$ was obtained by multiplying $T(1)$ for $k = 10$ by four.

We see that we obtained good efficiencies (above 81–89%) for the case $k - 6 \geqslant \log_2(P)$ even if we use a network of workstations as a parallel computing system. In [7], we also report on the results on a clustered network of workstations which is not as well adapted to our algorithm as the binary-tree structured network. There, the efficiency numbers were about 10–20% smaller for $P = 16$, which shows the importance of an optimized structure of the network.

[2] All further problems occurring in the parallelization of standard multigrid like the treatment of the coarse grid problems, where some agglomeration techniques have to be used to map these problems on a smaller amount of processors are not included in these estimates.

6. Conclusions

In this paper we presented new multilevel algorithms based on the generating system approach. Instead of level-oriented Gauss–Seidel iterations on the semidefinite system, which turn out to result just in standard multigrid algorithms, we used a point- and domain-oriented ordering of the unknowns. For these methods, the reduction rate is also independent of the grid size as for conventional multigrid methods but they can be interpreted as domain decomposition methods. The number of communication steps does not depend on the number of levels but depends only on the number of processors used. This makes these algorithms well suited for parallel systems with high setup times like for example workstation networks. Furthermore, if we take into account that the computational performance of the processors will increase faster than the performance of communication (hardware and software), parallel algorithms with a low number of communication steps will become more and more important in the future.

References

[1] J. Bramble, J. Pasciak and J. Xu, Parallel multilevel preconditioners, *Math. Comput.* **31** (1990) 333–390.
[2] A. George, Nested dissection of a regular finite element mesh, *SIAM J. Numer. Anal.* **10** (1973) 345–363.
[3] M. Griebel, Grid- and point-oriented multilevel algorithms, in: W. Hackbusch and G. Wittum, Eds., *Incomplete Decomposition (ILU): Theory, Technique and Application, Proc. 8th GAMM-Seminar, Kiel, 1992, Notes on Numerical Fluid Mechanics*, Vol. 41 (Vieweg Verlag, Braunschweig, 1992) 32–46.
[4] M. Griebel, Multilevel algorithms considered as iterative methods on semidefinite systems, *SIAM J. Sci. Comput.* **15** (1994) 547–565.
[5] M. Griebel, *Multilevel Methoden als Iterationsverfahren über Erzeugendensystemen*, Teubner Skr. Numer. (Teubner, Stuttgart, 1994).
[6] M. Griebel, parallel domain-oriented multilevel methods, *SIAM J. Sci. Comput.* **16** (1995) 1105–1125.
[7] M. Griebel and T. Neunhoeffer, Point- and domain-oriented multilevel algorithms—parallelization and implementational aspects, SFB-Report 342/18/94 A, TU München, Institut für. Informatik, 1994.
[8] M. Griebel and P. Oswald, On the abstract theory of additive and multiplicative schwarz methods, *Numer. Math.* **70** (1995) 163–180.
[9] R. Hempel and A. Schüller, Experiments with parallel multigrid algorithms using the SUPRENUM communications subroutine library, (Arbeitspapiere der GMD 141, GMD, 1988).
[10] O. McBryan, P. Fredericson, J. Linden, A. Schüller, K. Solchenbach, K. Stüben, C. Thole and U. Trottenberg, Multigrid methods on parallel computers—a survey of recent developments, *Impact Comput. Sci. Eng.* **3** (1991) 1–75.
[11] P. Oswald, On discrete norm estimates related to multilevel preconditioners in the finite element method, *Proc. Internat. Conf. Constr. Theory of Functions*, Varna, 1991.
[12] P. Oswald, Norm equivalencies and multilevel Schwarz preconditioning for variational problems (Bericht Math/92/1, FSU Jena, Mathematische Fakultät, 1992).
[13] J. Xu, Iterative methods by space decomposition and subspace correction: a unifying approach, *SIAM Rev.* **34** (1992) 581–613.
[14] X. Zhang, Multilevel Schwarz methods, *Numer. Math.* **63** (1992) 521–539.